Unicode is a standard for representing a much larger variety of characters beyond the roman alphabet (greek, russian, mathematical symbols, logograms from non-phonetic writing systems such as kanji, etc.)

In most circumstances, you will be able to use a unicode object just like a string.

If you encounter an error involving printing unicode, you can use the encode method to properly print the international characters, like this:

```
unicode_string = u"aaaÃ Ã§Ã§Ã§Ã±Ã±Ã±"
encoded_string = unicode_string.encode('utf-8')
print encoded_string
```

Getting Started

Once again: If you are new to Python, many students have recommended Google's Python class.

Problem 1: Get Twitter Data

As always, the first step is to make sure your assignment materials up to date.

To access the live stream, you will need to install the **oauth2 library** so you can properly authenticate.

This library is already installed on the class virtual machine, but you can install it yourself in your Python environment. (The command `$ pip install oauth2` should work for most environments.)

The steps below will help you set up your twitter account to be able to access the live 1% stream.

1. Create a twitter account if you do not already have one.
2. Go to **https://dev.twitter.com/apps** and log in with your twitter credentials.
3. Click "Create New App"
4. Fill out the form and agree to the terms. Put in a dummy website if you don't have one you want to use.
5. On the next page, click the "API Keys" tab along the top, then scroll all the way down until you see the section "Your Access Token"
6. Click the button "Create My Access Token". You can **Read more about Oauth authorization.**
7. You will now copy four values into twitterstream.py. These values are your "API Key", your "API secret", your "Access token" and your "Access token secret". All four should now be visible on the API Keys page. (You may see "API Key" referred to as "Consumer key" in some places in the code or on the web; they are synonyms.) Open twitterstream.py and set the variables corresponding to the api key, api secret, access token, and access secret. You will see code like the below:

   ```
   api_key = "<Enter api key>"
   api_secret = "<Enter api secret>"
   access_token_key = "<Enter your access token key here>"
   access_token_secret = "<Enter your access token secret here>"
   ```

8. Run the following and make sure you see data flowing and that no errors occur.

   ```
   $ python twitterstream.py > tweets.txt
   ```

   This command pipes the output to a file. Stop the program with Ctrl-C, but wait at least **3 minutes** for data to accumulate. Keep the file tweets.txt for the duration of the assignment; we will be reusing it in later problems. Don't use someone else's file; we will check for uniqueness in other parts of the assignment.

9. If you wish, modify the file to use the twitter search API to search for specific terms. For example, to search for the term "microsoft", you can pass the following url to the twitterreq function:

   ```
   https://api.twitter.com/1.1/search/tweets.json?q=microsoft
   ```

What to turn in: The first 20 lines of the twitter data you downloaded from the web. You can save the first 20 lines to a file `problem_1_submission.txt` by using the following command:

```
$ head -n 20 tweets.txt > problem_1_submission.txt
```

Problem 2: Derive the sentiment of each tweet

For this part, you will compute the sentiment of each tweet based on the sentiment scores of the terms in the tweet. The sentiment of a tweet is equivalent to the sum of the sentiment scores for each term in the tweet.

You are provided with a skeleton file `tweet_sentiment.py` which accepts two arguments on the command line: a *sentiment file* and a tweet file like the one you generated in Problem 1. You can run the skeleton program like this:

```
$ python tweet_sentiment.py Sentiment-Words.txt tweets.txt
```

The file Sentiment-Words.txt contains a list of pre-computed sentiment scores. Each line in the file contains a word or phrase followed by a sentiment score. Each word or phrase that is found in a tweet but not found in Sentiment-Words.txt should be given a sentiment score of 0. See the file AFINN-README.txt for more information.

To use the data in the Sentiment-Words.txt file, you may find it useful to build a dictionary. Note that the Sentiment-Words.txt file format is tab-delimited, meaning that the term and the score are separated by a tab character. A tab character can be identified a "\t".The following snippet may be useful:

```
afinnfile = open("Sentiment-Words.txt")
scores = {} # initialize an empty dictionary
for line in afinnfile:
  term, score  = line.split("\t")  # The file is tab-delimited. "\t" means "tab character"
  scores[term] = int(score)  # Convert the score to an integer.

print scores.items() # Print every (term, score) pair in the dictionary
```

The data in the tweet file you generated in Problem 1 is represented as *JSON*, which stands for JavaScript Object Notation. It is a simple format for representing nested structures of data --- lists of lists of dictionaries of lists of .... you get the idea.

Each line of `tweets.txt` represents a streaming message. Most, but not all, will be tweets. (The skeleton program will tell you how many lines are in the file.)

It is straightforward to convert a JSON string into a Python data structure; there is a library to do so called `json`.

To use this library, add the following to the top of `tweet_sentiment.py`

```
import json
```

Then, to parse the data in `tweets.txt`, you want to apply the function `json.loads` to every line in the file.

This function will parse the json data and return a python data stucture; in this case, it returns a dictionary. If needed, take a moment to **read the documentation for Python dictionaries**.

You can read the Twitter documentation to understand what information each tweet contains and how to access it, but it's not too difficult to deduce the structure by direct inspection.

Your script should print to stdout the sentiment of each tweet in the file, one numeric sentiment score per line. The first score should correspond to the first tweet, the second score should correspond to the second tweet, and so on. If you sort the scores, they won't match up. If you sort the tweets, they won't match up. If you put the tweets into a dictionary, the order will not be preserved. Once again: **The nth line of the file you submit should contain only a single number that represents the score of the nth tweet in the input file!**

NOTE: You must provide a score for **every** tweet in the sample file, even if that score is zero. You can assume the sample file will only include English tweets and no other types of streaming messages.

To grade your submission, we will run your program on a tweet file formatted the same way as the `tweets.txt` file you generated in Problem 1.

Hint: This is real-world data, and it can be messy! Refer to the twitter documentation to understand more about the data structure you are working with. Don't get discouraged, and ask for help on the forums if you get stuck!

What to turn in: The file `tweet_sentiment.py` after you've verified that it returns the correct answers.

Problem 3: Derive the sentiment of new terms

In this part you will be creating a script that computes the sentiment for the terms that **do not** appear in the file Sentiment-Words.txt.

Here's how you might think about the problem: We know we can use the sentiment-carrying words in Sentiment-Words.txt to deduce the overall sentiment of a tweet. Once you deduce the sentiment of a tweet, you can work backwards to deduce the sentiment of the non-sentiment carrying words that do *not* appear in Sentiment-Words.txt. For example, if the word `soccer` always appears in proximity with positive words like `great` and `fun`, then we can deduce that the term `soccer` itself carries a positive sentiment.

Don't feel obligated to use it, but the following paper may be helpful for developing a sentiment metric. Look at the Opinion Estimation subsection of the Text Analysis section in particular.

O'Connor, B., Balasubramanyan, R., Routedge, B., & Smith, N. From Tweets to Polls: Linking Text Sentiment to Public Opinion Time Series. (ICWSM), May 2010.

You are provided with a skeleton file `term_sentiment.py` which accepts the same two arguments as tweet_sentiment.py and can be executed using the following command:

```
$ python term_sentiment.py Sentiment-Words.txt tweets.txt
```

Your script should print output to stdout. Each line of output should contain a term, followed by a space, followed by the sentiment. That is, each line should be in the format `<term:string> <sentiment:float>`

For example, if you have the pair `("foo", 103.256)` in Python, it should appear in the output as:

```
foo 103.256
```

The order of your output does not matter.

What to turn in: The file `term_sentiment.py`

How we will grade Part 3: We will run your script on a file that contains strongly positive and strongly negative tweets and verify that the non-sentiment-carrying terms in the strongly positive tweets are assigned a higher score than the non-sentiment-carrying terms in negative tweets.Â Your scores need not (and likely will not) exactly match any specific solution.

If the grader is returning "Formatting error: ", make note of the line of text returned in the message. This line corresponds to a line of your output. The grader will generate this error if `line.split()` does not return **exactly two items**. One common source of this error is to not remove the two calls to the "lines" function in the solution template; this function prints the number of lines in each file. Make sure to check the first two lines of your output!